

Chapter Overview

Search

- ❖ **Motivation**
- ❖ **Objectives**
- ❖ **Search as Problem-Solving**
 - ❖ problem formulation
 - ❖ problem types
- ❖ **Uninformed Search**
 - ❖ breadth-first
 - ❖ depth-first
 - ❖ uniform-cost search
 - ❖ depth-limited search
 - ❖ iterative deepening
 - ❖ bi-directional search
- ❖ **Informed Search**
 - ❖ best-first search
 - ❖ search with heuristics
 - ❖ memory-bounded search
 - ❖ iterative improvement search
- ❖ **Non-Traditional Search**
 - ❖ local search and optimization
 - ❖ constraint satisfaction
 - ❖ search in continuous spaces
 - ❖ partially observable worlds
- ❖ **Important Concepts and Terms**
- ❖ **Chapter Summary**

Logistics

- ❖ see also [Moodle Discussion Board Announcement](#)
- ❖ **Labs**
 - ❖ L2 demos in lab continued
 - ❖ L3 demos in lab starting Tue, April 28
 - ❖ L4: Search Algorithms
 - ❖ continuation of L3: uniform cost, best-first (greedy), A*
 - ❖ available now
 - ❖ deadline moved to Tue, May 19
- ❖ **Quizzes**
 - ❖ Q3 deadline tonight
 - ❖ Q4 open now, deadline next Tue (April 28)
- ❖ **Project**
 - ❖ evaluation in progress of the following documentation parts
 - ❖ Overview
 - ❖ Features, Requirements, Evaluation Criteria

Non-Traditional Search

local search and optimization
constraint satisfaction
search in continuous spaces
partially observable worlds

Local Search and Optimization

- ◆ for some problem classes, it is sufficient to find a solution
 - ◆ the path to the solution is not relevant
- ◆ memory requirements can be dramatically relaxed by modifying the current state
 - ◆ only information about the current state is kept
 - ◆ all previous states are discarded
 - ◆ may have serious consequences
 - ◆ no path information kept
 - ◆ impacts completeness, optimality
 - ◆ since only information about the current state is kept, such methods are called *local*

Iterative Improvement Search

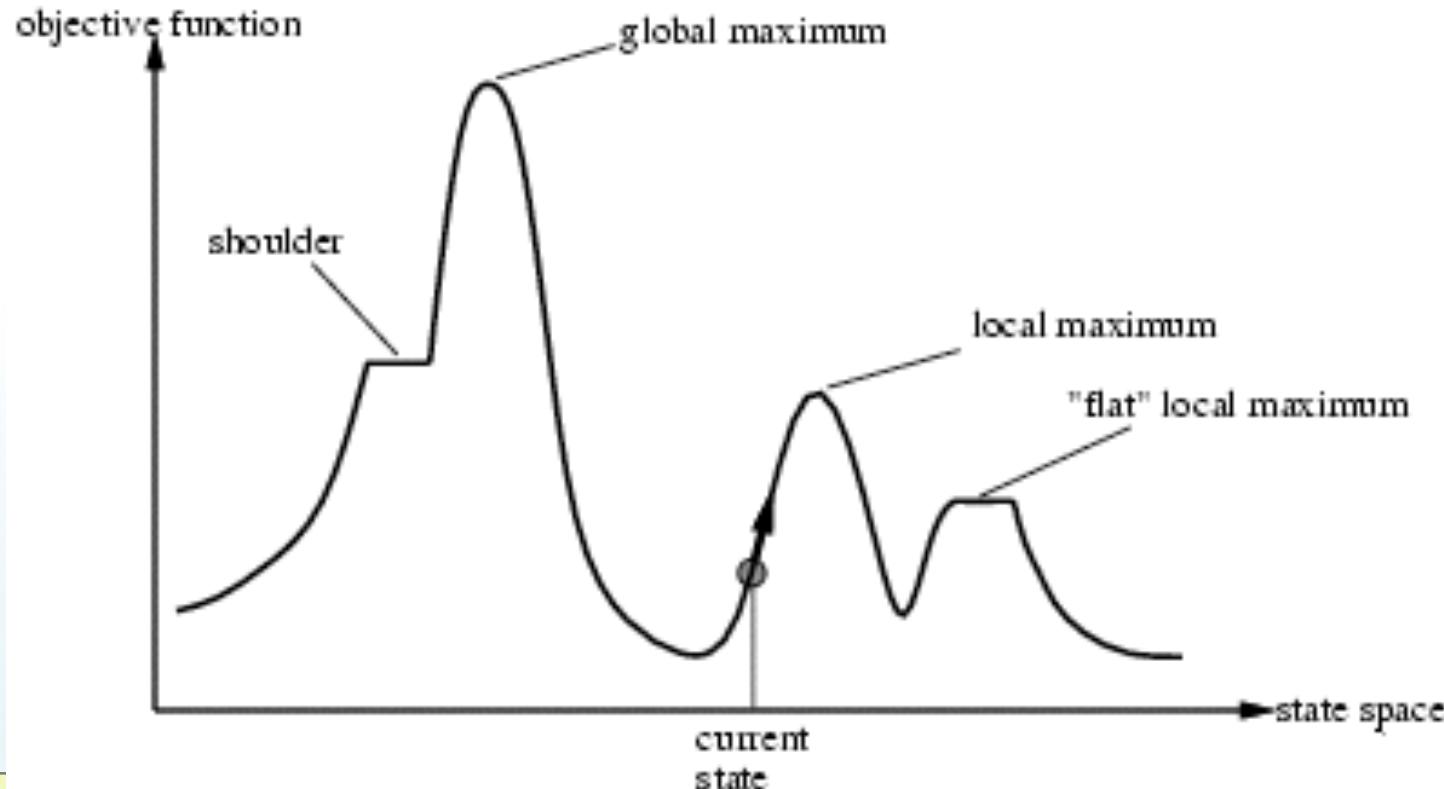
- ◆ for some problems, the state description provides all the information required for a solution
 - ◆ path to the goal is not critical
 - ◆ path costs become irrelevant
 - ◆ global maximum or minimum corresponds to the optimal solution
 - ◆ local optimum may be “good enough”
- ◆ iterative improvement algorithms start with some configuration, and try modifications to improve the quality
 - ◆ 8-queens: number of un-attacked queens
 - ◆ VLSI layout: total wire length
- ◆ analogy: state space as landscape with hills and valleys

Hill-Climbing Search

- ❖ **continually moves uphill**
 - ❖ increasing value of the evaluation function
 - ❖ gradient descent search is a variation that moves downhill
- ❖ **very simple strategy with low space requirements**
 - ❖ stores only the state and its evaluation, no search tree
- ❖ **problems**
 - ❖ local maxima
 - ❖ algorithm can't go higher, but is not at a satisfactory solution
 - ❖ plateau
 - ❖ area where the evaluation function is flat
 - ❖ ridges
 - ❖ search may oscillate slowly

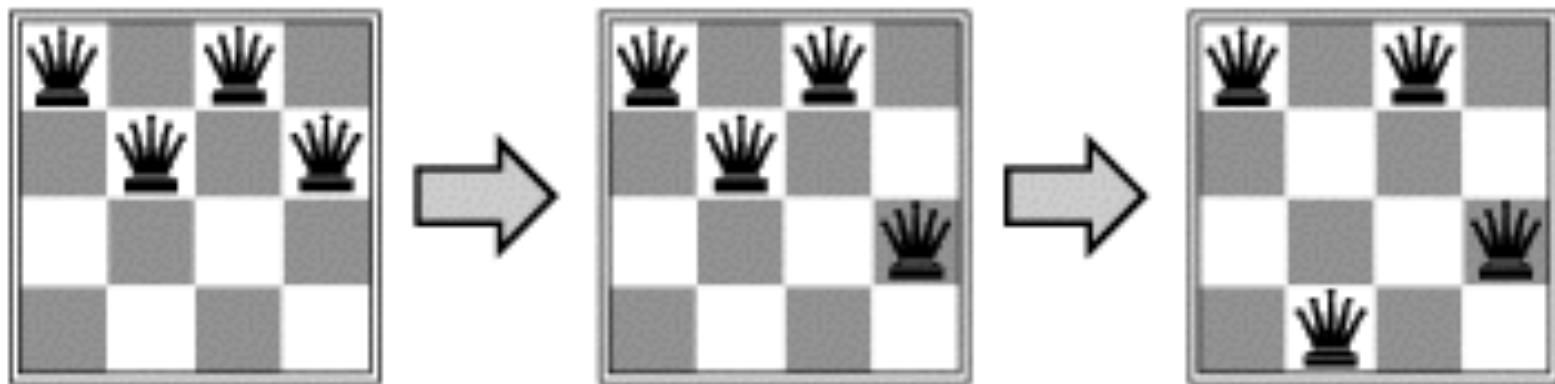
Hill-climbing search

- ❖ Problem: depending on initial state, can get stuck in local maxima



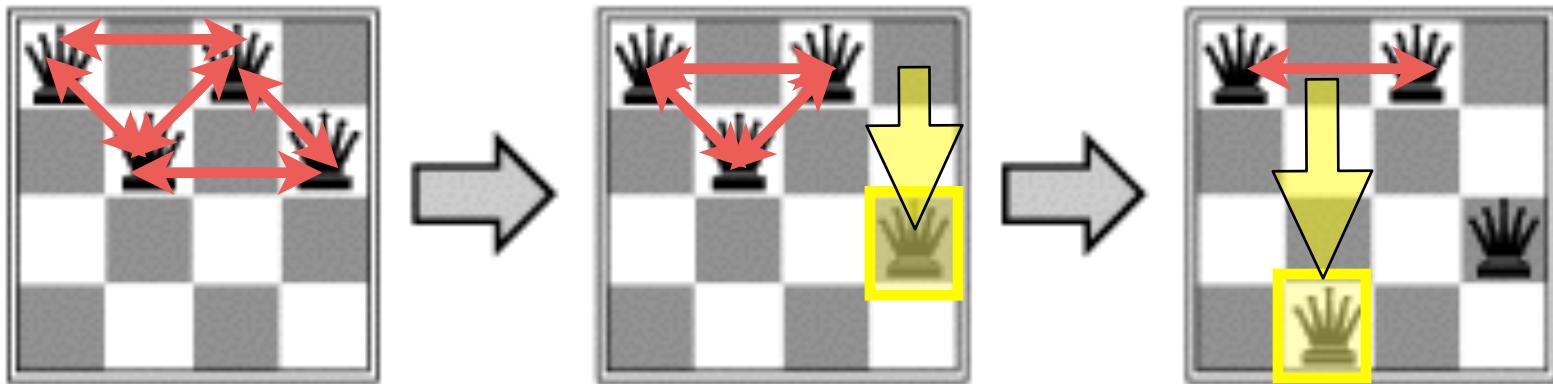
Example: n -queens

- ❖ Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



Hill-climbing: 4-queens

- ❖ 4 queens on a 4x4 board
- ❖ optimization criterion: number of attacked pairs of queens
 - ❖ illustrated through red lines



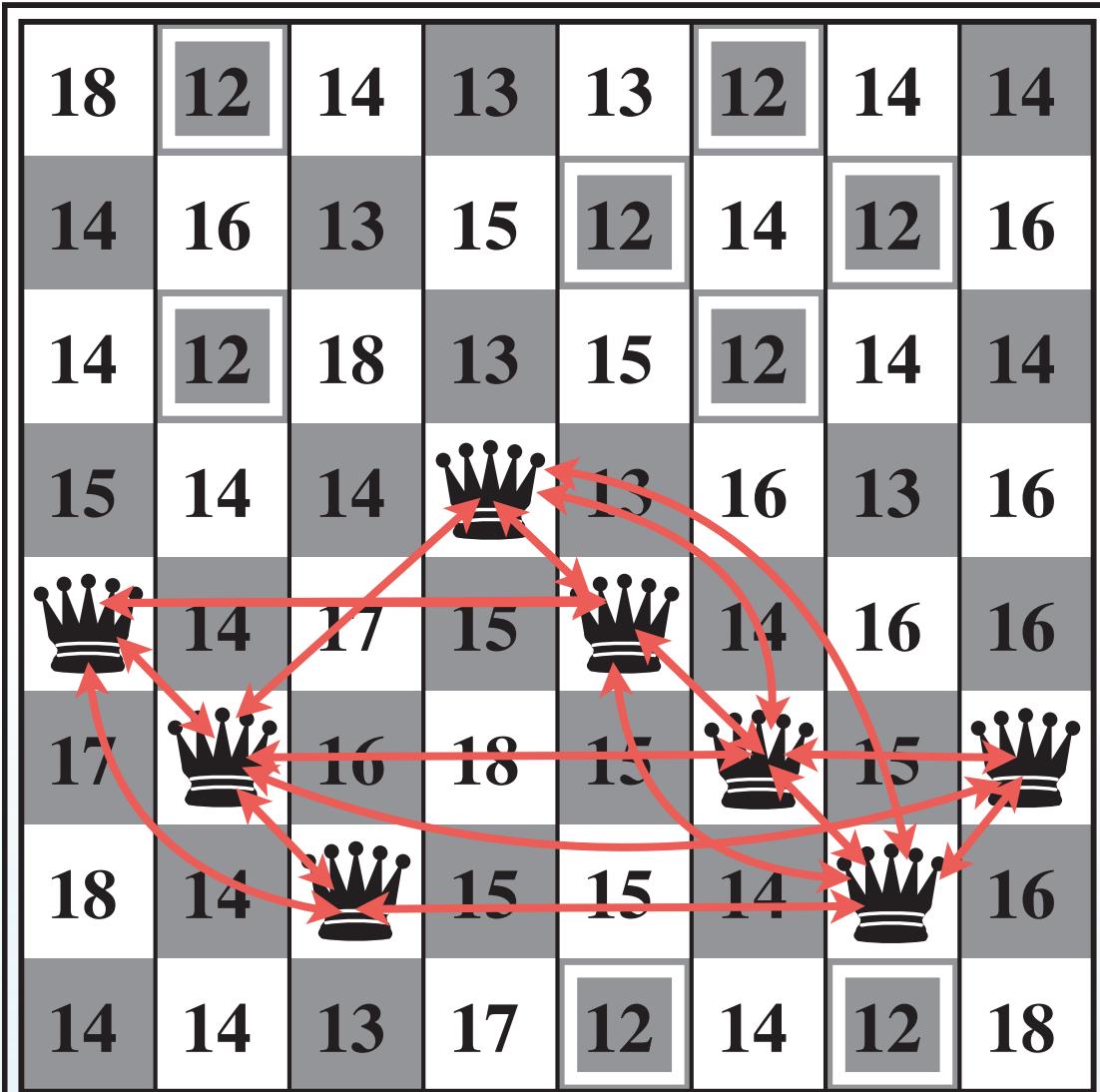
Hill-climbing search: 8-queens

- ❖ $h = \text{number of pairs of queens that are attacking each other}$
- ❖ $h = 17$ for the above state
- ❖ best moves are marked ($h=12$)

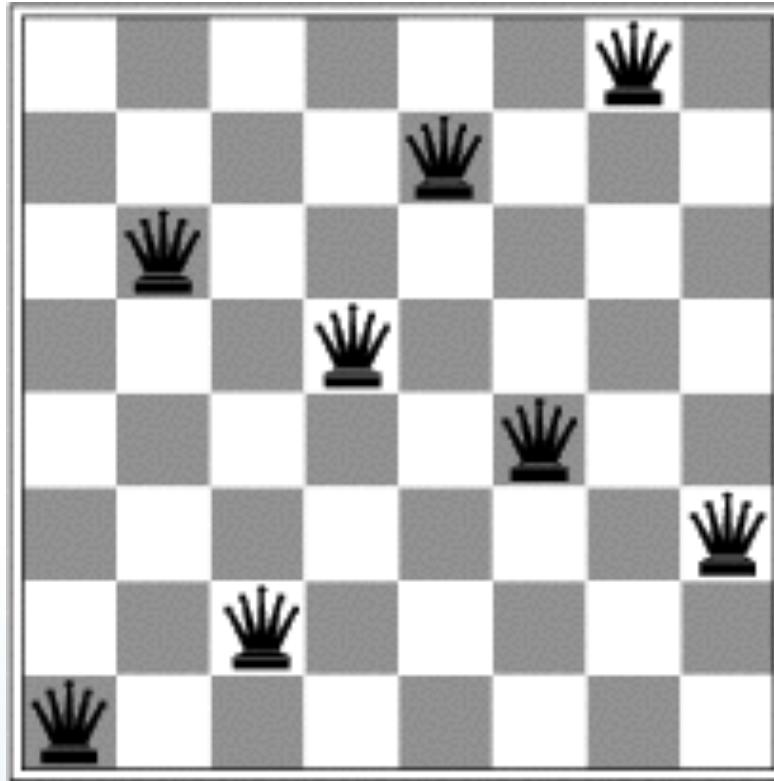
18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	crown	13	16	13	16
crown	14	17	15	crown	14	16	16
17	crown	16	18	15	crown	15	crown
18	14	crown	15	15	14	crown	16
14	14	13	17	12	14	12	18

8-queens: Attacked Pairs

- ❖ $h = \text{number of pairs of queens that are attacking each other}$
- ❖ $h = 17$ for the above state
- ❖ best moves are marked ($h=12$)

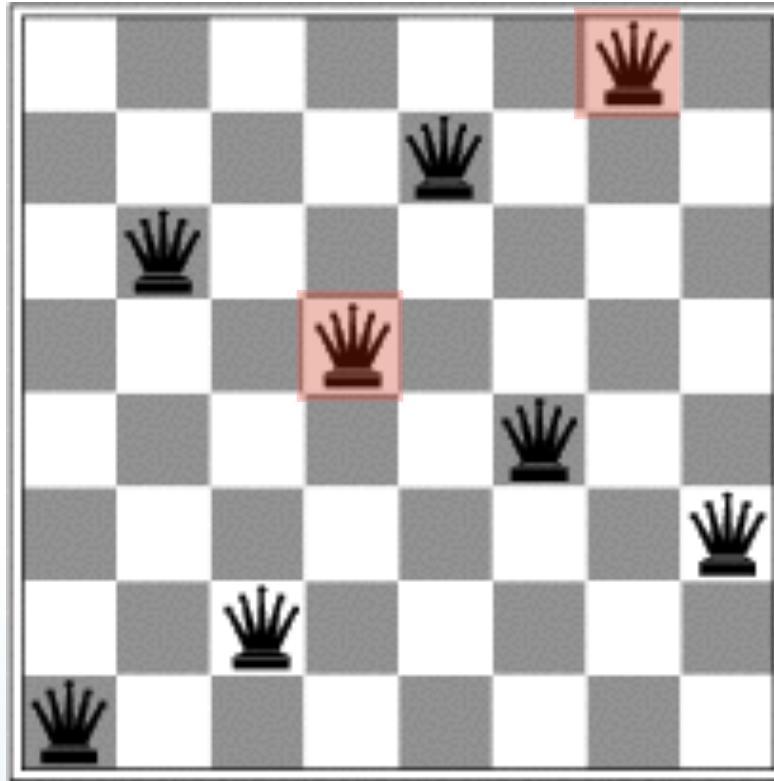


Hill-climbing search: 8-queens problem



- ❖ A local minimum with $h = 1$

Hill-climbing search: 8-queens problem



- ❖ A local minimum with $h = 1$
 - ❖ one pair of queens attacking each other

Simulated Annealing

- ◆ similar to hill-climbing, but some down-hill movement
 - ◆ random move instead of the best move
 - ◆ depends on two parameters
 - ❖ ΔE , energy difference between moves; T, temperature
 - ❖ temperature is slowly lowered, making bad moves less likely
- ◆ analogy to annealing
 - ◆ gradual cooling of a liquid until it freezes
- ◆ will find the global optimum if the temperature is lowered slowly enough
- ◆ applied to routing and scheduling problems
 - ◆ VLSI layout, scheduling

Local Beam Search

- ◆ **variation of beam search**
 - ◆ a path-based method that looks at several paths “around” the current one
- ◆ **keeps k states in memory, instead of only one**
 - ◆ information between the states can be shared
 - ❖ moves to the most promising areas
- ◆ **stochastic local beam search selects the k successor states randomly**
 - ◆ with a probability determined by the evaluation function

Genetic Algorithms (GAs)

◆ variation of stochastic beam search

- ◆ successor states are generated as variations of *two* parent states, not only one
- ◆ corresponds to natural selection with sexual reproduction
 - ◆ with a twist: swapping body parts
- ◆ mutation provides an additional random element
 - ❖ random modification of features (variable values)

GA Terminology

◆ population

- ◆ set of k randomly generated states

◆ generation

- ◆ population at a point in time
- ◆ usually, propagation is synchronized for the whole population

◆ individual

- ◆ one element from the population
- ◆ described as a string over a finite alphabet
 - ❖ binary, ACGT, letters, digits
 - ❖ consistent for the whole population

◆ fitness function

- ◆ evaluation function in search terminology
- ◆ higher values lead to better chances for reproduction

GA Principles

◆ reproduction

- ◆ the state description of the two parents is split at the crossover point
 - ❖ determined in advance, often randomly chosen
 - ❖ must be the same for both parents
- ◆ one part is combined with the other part of the other parent
 - ❖ one or both of the descendants may be added to the population
 - ❖ compatible state descriptions should assure viable descendants
 - ❖ depends on the choice of the representation
 - ❖ may not have a high fitness value

◆ mutation

- ◆ each individual may be subject to random modifications in its state description
 - ❖ usually with a low probability

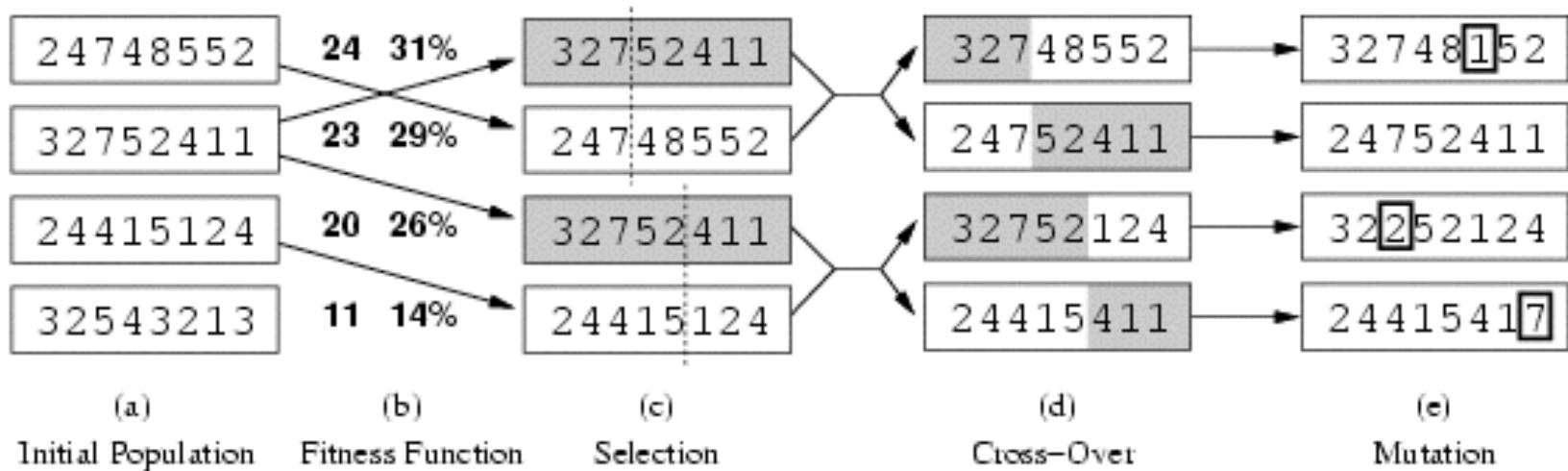
◆ schema

- ◆ useful components of a solution can be preserved across generations

GA Applications

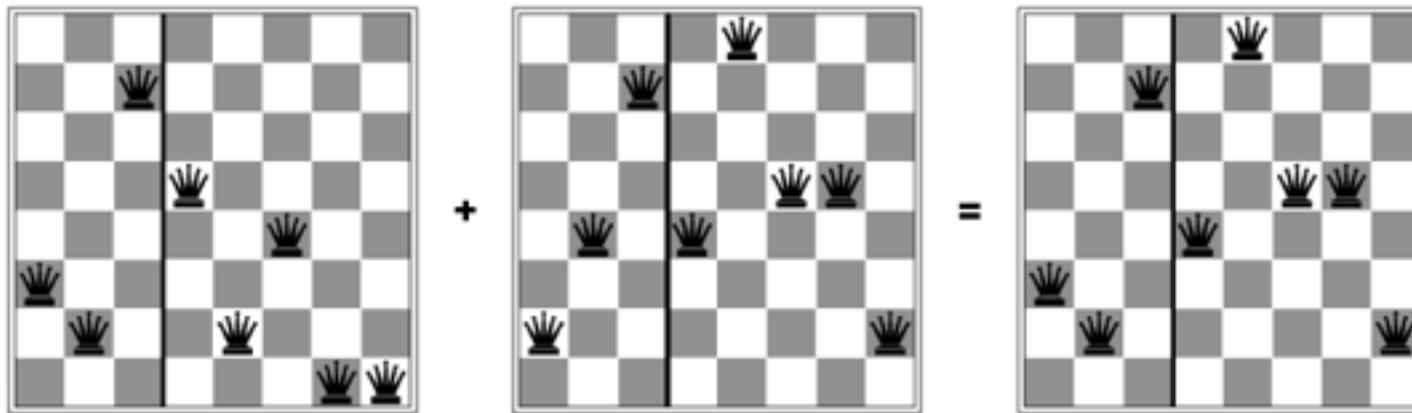
- ◆ often used for optimization problems
 - ◆ circuit layout, system design, scheduling
- ◆ termination
 - ◆ “good enough” solution found
 - ◆ no significant improvements over several generations
 - ◆ time limit

Example: Genetic algorithm for N-Queens



- ❖ **Fitness function: number of non-attacking pairs of queens (min = 0, max = $8 \times 7/2 = 28$)**
- ❖ $24/(24+23+20+11) = 31\%$
- ❖ $23/(24+23+20+11) = 29\% \text{ etc}$

Genetic Algorithm for N-Queens



Constraint Satisfaction

CSP terms and definitions

Examples

Map Coloring

Cryptarithmetic

N-Queens

CSP and Search

Heuristics

Analyzing Constraints

Local Search and CSP

CSP Definitions

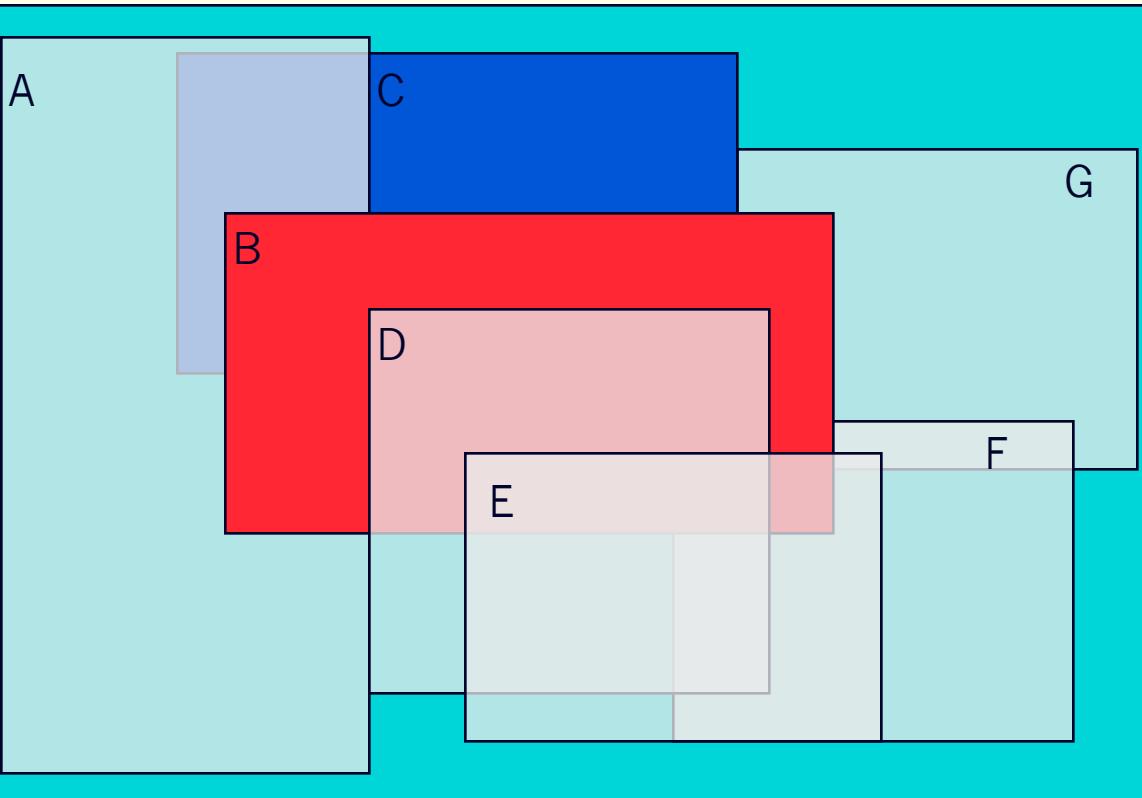
- ❖ **satisfies additional structural properties of the problem**
 - ❖ may depend on the representation of the problem
- ❖ **the problem is defined through a set of domain variables**
 - ❖ variables can have possible values specified by the problem
 - ❖ constraints describe allowable combinations of values for a subset of the variables
- ❖ **state in a CSP**
 - ❖ defined by an assignment of values to some or all variables
- ❖ **consistent assignment**
 - ❖ does not violate any constraints
 - ❖ also called *legal* assignment
- ❖ **complete assignment**
 - ❖ every variable is mentioned
- ❖ **solution to a CSP**
 - ❖ complete assignment that satisfies all constraints
 - ❖ solutions may be ranked according to an objective function

CSP Approach

- ◆ the goal test is decomposed into a set of constraints on variables
 - ◆ checks for violation of constraints before new nodes are generated
 - ❖ must backtrack if constraints are violated
 - ◆ forward-checking looks ahead to detect unsolvability
 - ❖ based on the current values of constraint variables

CSP Example: Map Coloring

- ◆ color a map with three colors so that adjacent countries have different colors



variables (regions):

A, B, C, D, E, F, G

values (colors):

$\{red, green, blue\}$

constraints:

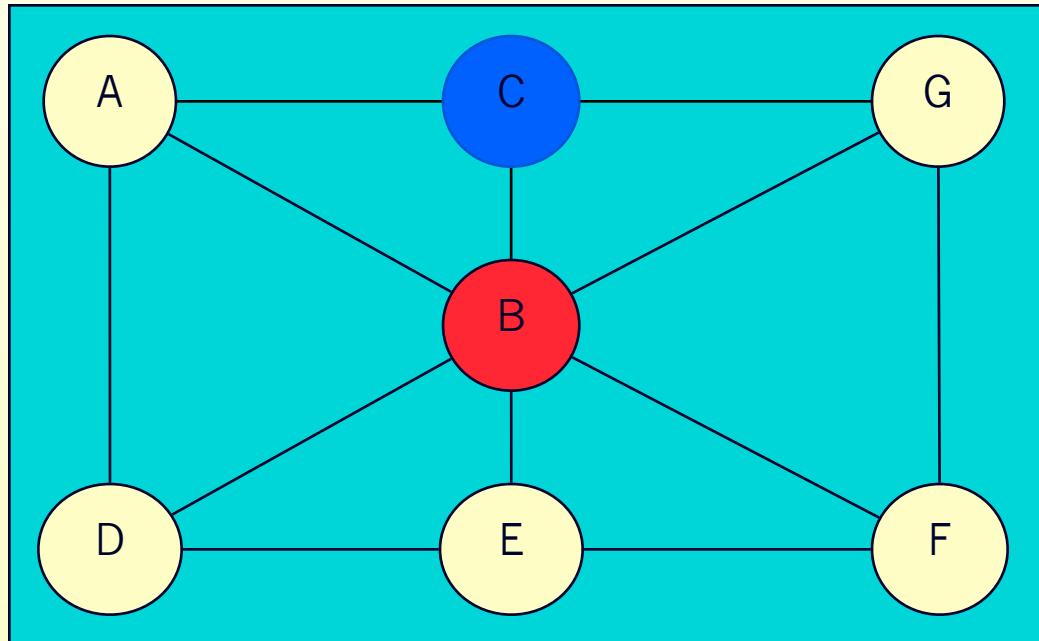
“no neighboring regions have the same color”

legal combinations for A, B:

$\{(red, \cancel{green}), (red, \cancel{blue}),$
 $(green, red), (green, \cancel{blue}),$
 $(blue, red), (blue, \cancel{green})\}$

Constraint Graph

- ◆ visual representation of a CSP
 - ◆ variables are nodes
 - ◆ arcs are constraints



the map coloring example
represented as constraint
graph

Example: Map-Coloring



- ❖ Variables WA, NT, Q, NSW, V, SA, T
- ❖ Domains $D_i = \{\text{red}, \text{green}, \text{blue}\}$
- ❖ Constraints: adjacent regions must have different colors
- ❖ e.g., $\text{WA} \neq \text{NT}$, or $(\text{WA}, \text{NT}) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), (\text{green}, \text{red}), (\text{green}, \text{blue}), (\text{blue}, \text{red}), (\text{blue}, \text{green})\}$

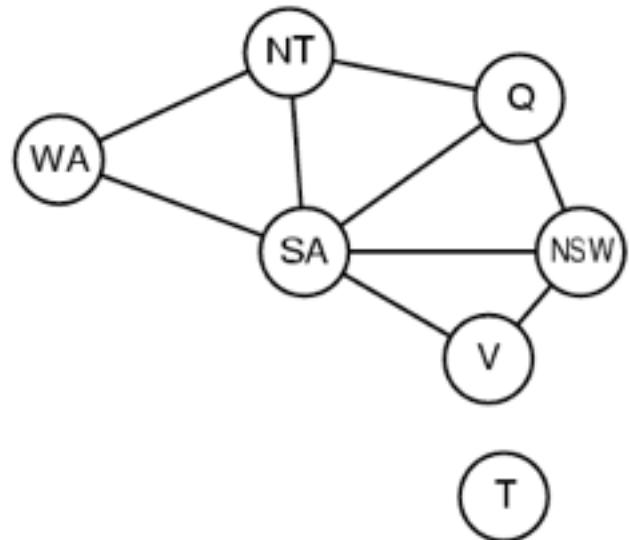
Example: Australia Map-Coloring



- ❖ Solutions are **complete** and **consistent** assignments, e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

Constraint graph

- ❖ **Binary CSP:** each constraint relates two variables
 - ❖ arcs connect exactly two nodes
 - ❖ n-ary CSP uses hypergraphs where arcs can connect more than two nodes
- ❖ **Constraint graph:** nodes are variables, arcs are constraints



Varieties of CSPs

❖ Discrete variables

- ❖ finite domains:
 - ❖ n variables, domain size d $\asymp O(d^n)$ complete assignments
 - ❖ e.g., Boolean CSPs, incl. \sim Boolean satisfiability (NP-complete)
- ❖ infinite domains:
 - ❖ integers, strings, etc.
 - ❖ e.g., job scheduling, variables are start/end days for each job
 - ❖ need a constraint language, e.g., $\text{StartJob1} + 5 \leq \text{StartJob3}$

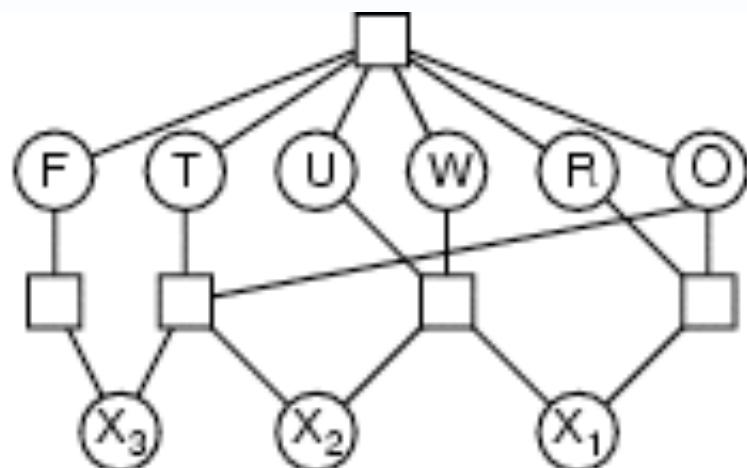
❖ Continuous variables

- ❖ e.g., start/end times for Hubble Space Telescope observations
- ❖ linear constraints solvable in polynomial time by linear programming

Example: Cryptarithmetic

- ❖ **Variables:** F T U W R O X₁ X₂ X₃
- ❖ **Domains:** {0,1,2,3,4,5,6,7,8,9}
- ❖ **Constraints:** Alldiff (F,T,U,W,R,O)
 - ❖ O + O = R + 10 · X₁
 - ❖ X₁ + W + W = U + 10 · X₂
 - ❖ X₂ + T + T = O + 10 · X₃
 - ❖ X₃ = F, T ≠ 0, F ≠ 0

$$\begin{array}{r} \text{T} \quad \text{W} \quad \text{O} \\ + \quad \text{T} \quad \text{W} \quad \text{O} \\ \hline \text{F} \quad \text{O} \quad \text{U} \quad \text{R} \end{array}$$



Benefits of CSP

- ◆ **standardized representation pattern**
 - ◆ variables with assigned values
 - ◆ constraints on the values
 - ◆ allows the use of generic heuristics
 - ❖ no domain knowledge is required

CSP as Incremental Search Problem

◆ initial state

- ◆ all (or at least some) variables unassigned

◆ successor function

- ◆ assign a value to an unassigned variable
- ◆ must not conflict with previously assigned variables

◆ goal test

- ◆ all variables have values assigned
- ◆ no conflicts possible
 - ❖ not allowed in the successor function

◆ path cost

- ◆ e.g. a constant for each step
- ◆ may be problem-specific

CSPs and Search

- ◆ in principle, any search algorithm can be used to solve a CSP
 - ◆ awful branching factor
 - ❖ $n*d$ for n variables with d values at the top level, $(n-1)*d$ at the next level, etc.
 - ◆ not very efficient, since they neglect some CSP properties
 - ❖ commutativity: the order in which values are assigned to variables is irrelevant, since the outcome is the same

Backtracking Search for CSPs

- ◆ a variation of depth-first search that is often used for CSPs
 - ◆ values are chosen for one variable at a time
 - ◆ if no legal values are left, the algorithm backs up and changes a previous assignment
 - ◆ very easy to implement
 - ❖ initial state, successor function, goal test are standardized
 - ◆ not very efficient
 - ❖ can be improved by trying to select more suitable unassigned variables first

Improving backtracking efficiency

- ❖ General-purpose methods can give huge gains in speed:
 - ❖ Which variable should be assigned next?
 - ❖ In what order should its values be tried?
 - ❖ Can we detect inevitable failure early?

Heuristics for CSP

- ◆ **most-constrained variable (minimum remaining values, “fail-first”)**
 - ◆ variable with the fewest possible values is selected
 - ◆ tends to minimize the branching factor
- ◆ **most-constraining variable**
 - ◆ variable with the largest number of constraints on other unassigned variables
- ◆ **least-constraining value**
 - ◆ for a selected variable, choose the value that leaves more freedom for future choices

Analyzing Constraints

◆ forward checking

- ◆ when a value X is assigned to a variable, inconsistent values are eliminated for all variables connected to X
 - ❖ identifies “dead” branches of the tree before they are visited

◆ constraint propagation

- ◆ analyses interdependencies between variable assignments via *arc consistency*
 - ❖ an arc between X and Y is consistent if for every possible value x of X , there is some value y of Y that is consistent with x
 - ❖ more powerful than forward checking, but still reasonably efficient
 - ❖ but does not reveal every possible inconsistency

Local Search and CSP

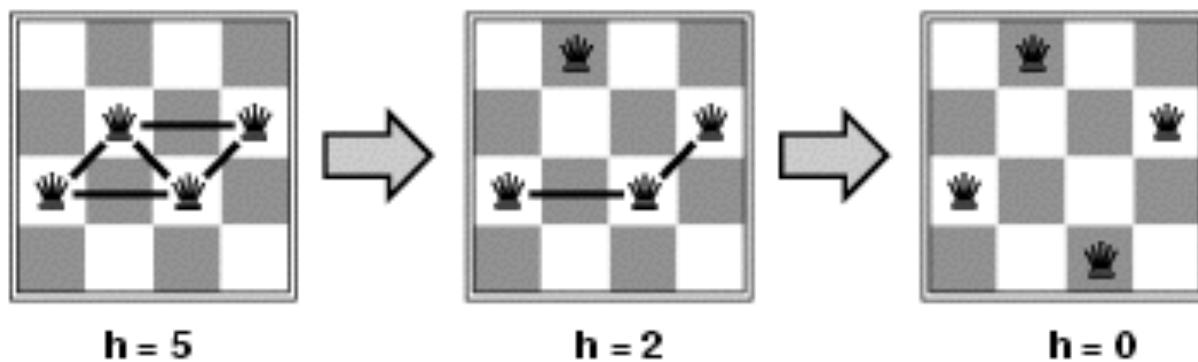
- ◆ local search (iterative improvement) is frequently used for constraint satisfaction problems
 - ◆ values are assigned to all variables
 - ◆ modification operators move the configuration towards a solution
- ◆ often called heuristic repair methods
 - ◆ repair inconsistencies in the current configuration
- ◆ simple strategy: min-conflicts
 - ◆ minimizes the number of conflicts with other variables
 - ◆ solves many problems very quickly
 - ❖ million-queens problem in less than 50 steps
- ◆ can be run as *online* algorithm
 - ◆ use the current state as new initial state

Local search for CSPs

- ❖ Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- ❖ To apply to CSPs:
 - ❖ allow states with unsatisfied constraints
 - ❖ operators **reassign** variable values
- ❖ Variable selection: randomly select any conflicted variable
- ❖ Value selection by **min-conflicts** heuristic:
 - ❖ choose value that violates the fewest constraints
 - ❖ i.e., hill-climb with $h(n)$ = total number of violated constraints

Example: 4-Queens

- ❖ **States:** 4 queens in 4 columns ($4^4 = 256$ states)
- ❖ **Actions:** move queen in column
- ❖ **Goal test:** no attacks
- ❖ **Evaluation:** $h(n) = \text{number of attacks}$



- ❖ Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)

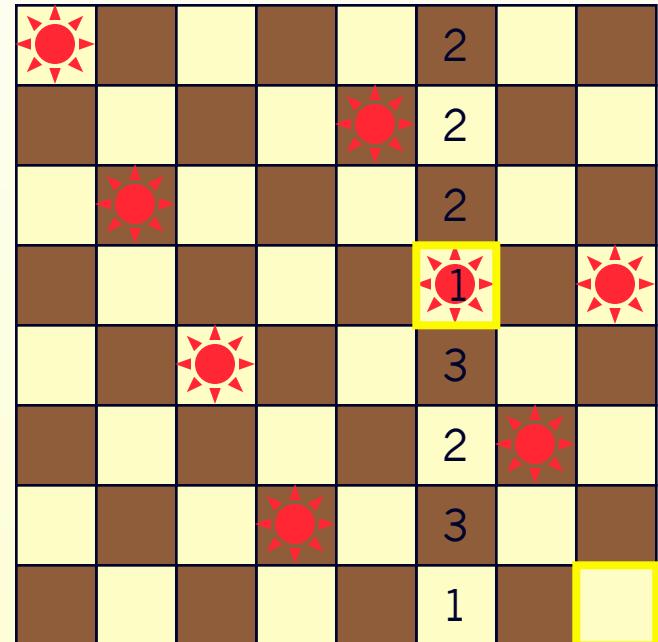
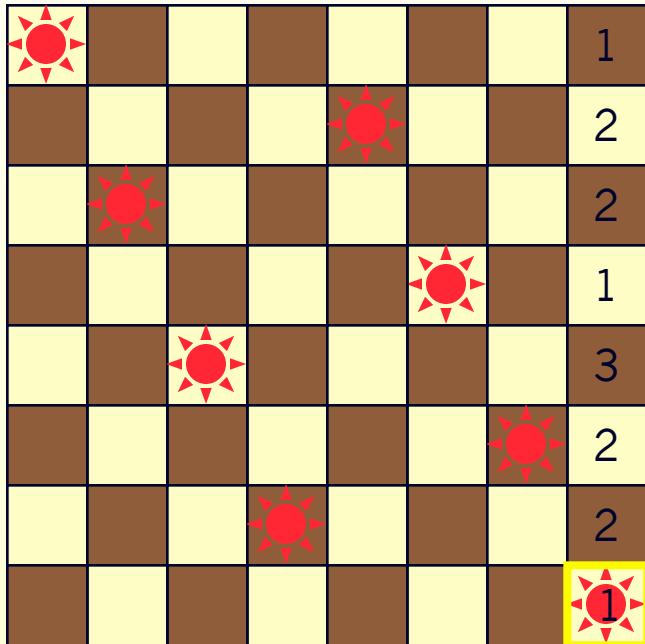
Analyzing Problem Structures

- ◆ some problem properties can be derived from the structure of the respective constraint graph
 - ◆ isolated sub-problems
 - ❖ no connections to other parts of the graph
 - ❖ can be solved independently
 - ❖ e.g. “islands” in map-coloring problems
 - ❖ dividing a problem into independent sub-problems reduces complexity tremendously
 - ❖ ideally from exponential to polynomial or even linear
 - ◆ tree
 - ❖ if the constraint graph is a tree, the CSP can be solved in time linear in the number of variables
 - ❖ sometimes solutions can be found by reducing a general graph to a tree
 - ❖ nodes are removed or collapsed

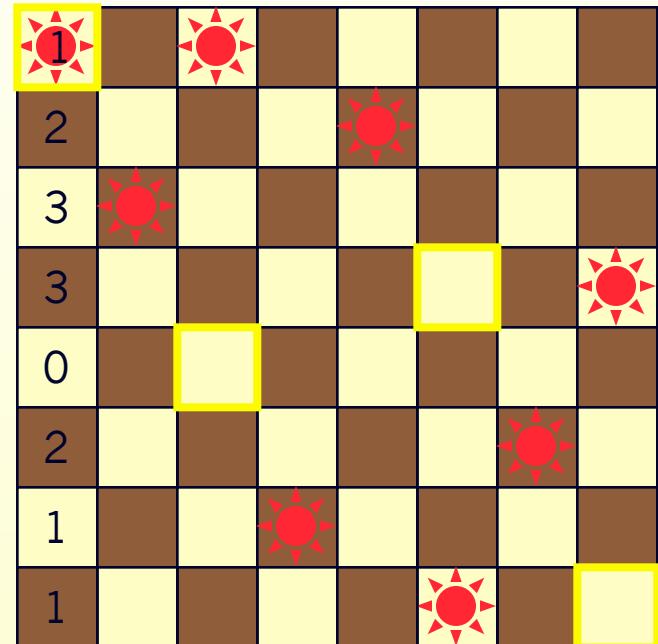
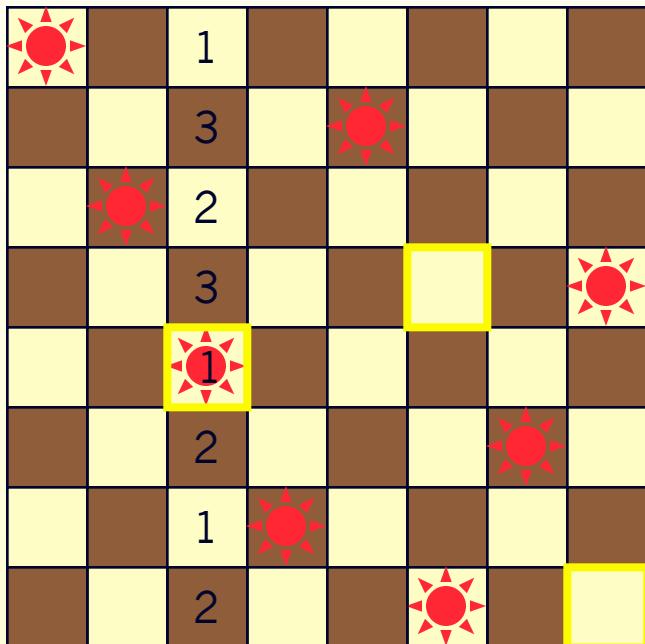
8-Queens with Min-Conflicts

- ◆ one queen in each column
 - ◆ usually several conflicts
- ◆ calculate the number of conflicts for each possible position of a selected queen
- ◆ move the queen to the position with the least conflicts

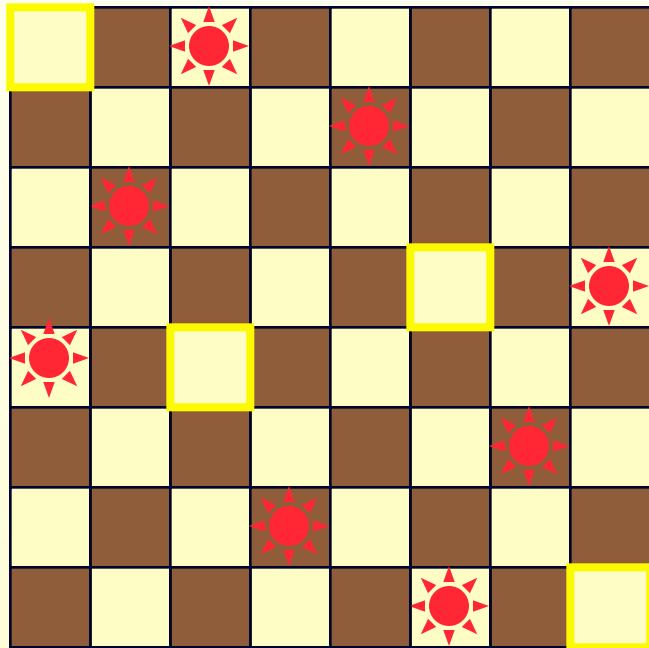
8-Queens Example 1



8-Queens Example 1

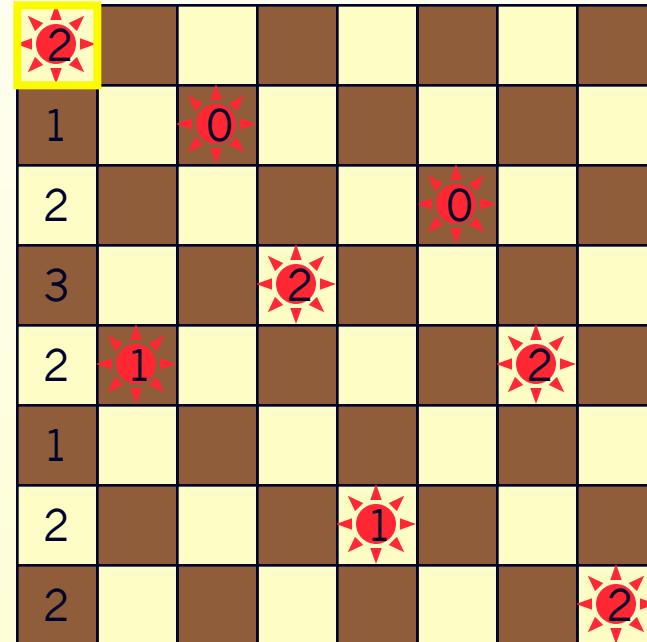
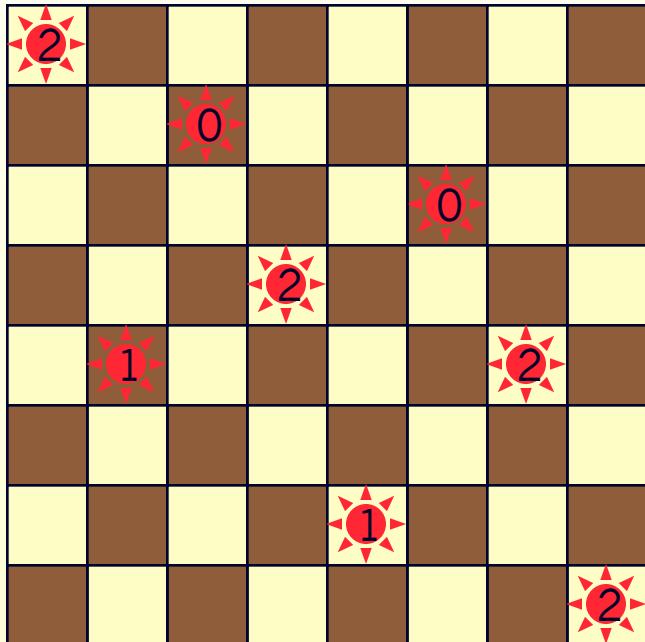


8-Queens Example 1

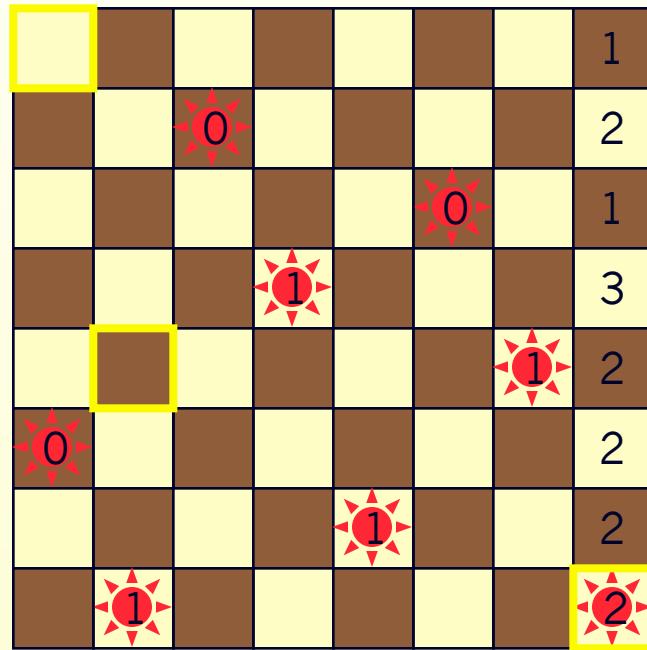
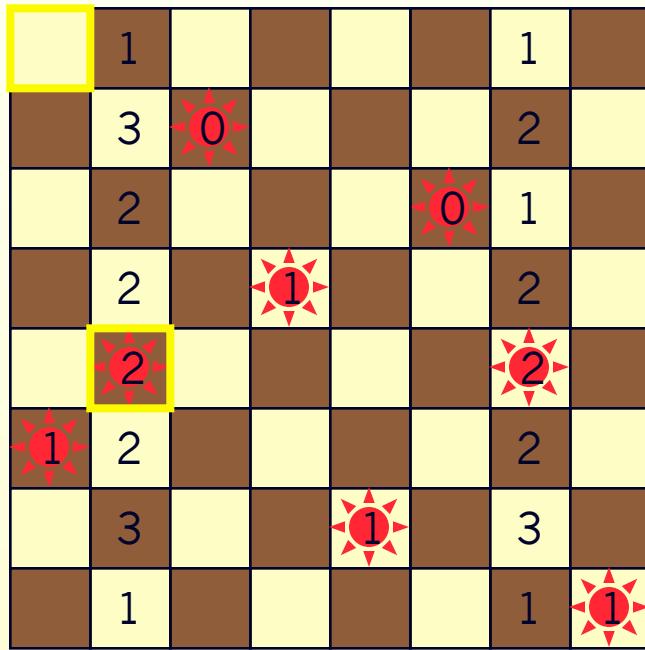


- ❖ solution found in 4 steps
- ❖ min-conflicts heuristic
- ❖ uses additional heuristics to select the “best” queen to move
 - ❖ try to move out of the corners
 - ❖ similar to least-constraining value heuristics

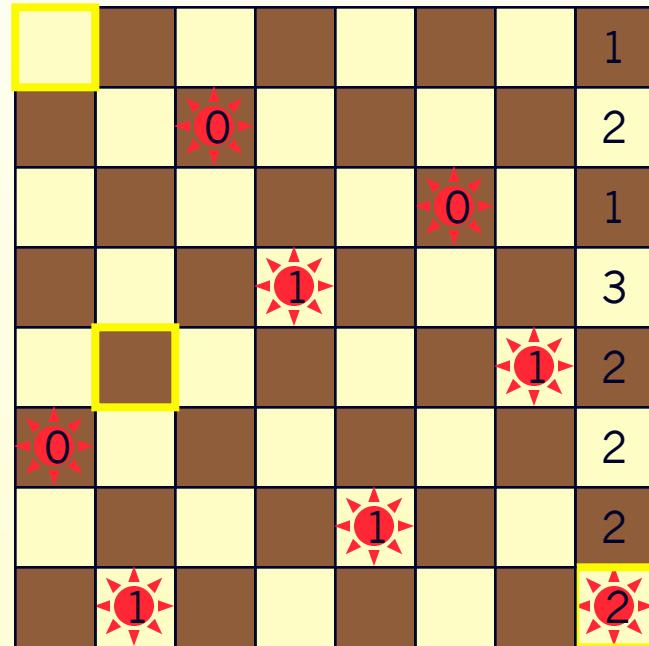
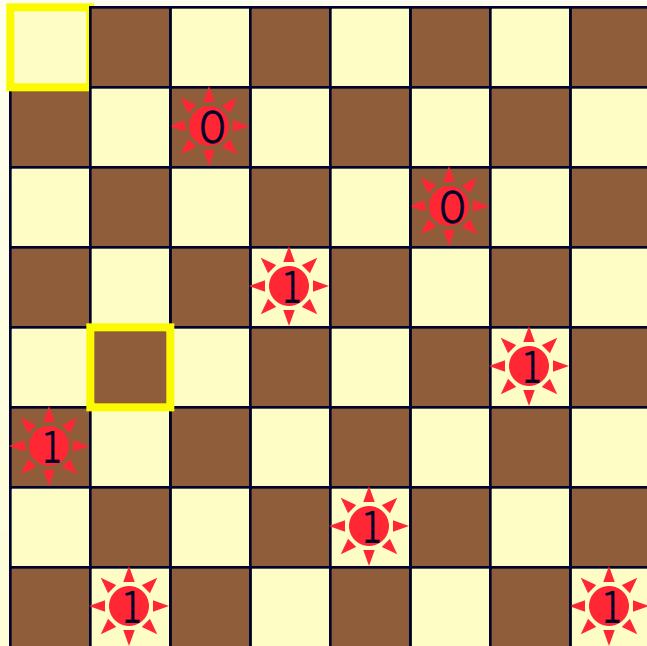
8-Queens Example 2



8-Queens Example 2



8-Queens Example 2



CSP Properties

- ❖ **discrete variables over finite domains**
 - ❖ relatively simple
 - ❖ e.g. map coloring, 8-queens
 - ❖ number of variable assignments can be d^n
 - ❖ domain size d, n variables
 - ❖ exponential time complexity (worst-case)
 - ❖ in practice, generic CSP algorithms can solve some problems much larger than regular search algorithms
- ❖ **more complex problems may require the use of a constraint language**
 - ❖ it is not possible to enumerate all combinations of values
 - ❖ e.g. job scheduling (“precedes”, “duration”)
- ❖ **related problems are studied in the field of Operations Research**
 - ❖ often continuous domains; e.g. linear programming

Search in Continuous Spaces

- ❖ almost all algorithms discussed earlier are suited only for discrete spaces
 - ❖ some variations of hill climbing and simulated annealing work for continuous environments as well
- ❖ continuous state and action spaces have infinite branching factors
 - ❖ really bad for most conventional search algorithms
- ❖ many techniques for search in continuous spaces have been developed in other fields
 - ❖ the development of calculus by Newton and Leibniz provided the main tools

Gradient Descent Search

- ❖ from the current state, select the direction with the steepest slope
 - ❖ the gradient of the objective function gives the magnitude and direction of the steepest slope
 - ❖ finding the maximum corresponds to solving an equation
 - ❖ in the landscape analogy, the mathematical maximum is often called a minimum since it is the lowest point
 - ❖ in many cases, it is not practical to calculate the *global* maximum
 - ❖ but it is often easy to compute the *local* maximum
- ❖ can be viewed as the inverse of hill climbing

Search with Non-Deterministic Actions

- ❖ most of the earlier search algorithms assume an environment that is fully observable and deterministic
 - ❖ this allows off-line search
 - ❖ the agent can first do the calculations for the search until it finds the goal, and then pursue a particular path by executing the respective actions
- ❖ in non-deterministic environments, the agent needs to deal with contingencies
 - ❖ situations where important information is only available at the time the agent executes its actions
 - ❖ the solution to a problem then is not a sequence of actions, but a *contingency plan (strategy)*
 - ❖ it can contain nested if-then-else statements

AND-OR Search Trees

- ❖ trees used to find contingent solutions for non-deterministic problems
- ❖ **OR nodes express choices the agent has in each state**
 - ❖ these correspond to the nodes in a deterministic search tree
- ❖ **AND nodes reflect choices by the environment (contingencies)**
 - ❖ the agent needs to prepare a plan for all potential choices, which corresponds to a set of AND-connected nodes

Partially Observable Environments

- ❖ **searching with no observation**
 - ❖ sensor-less or conformant problems
- ❖ **partial observations and percepts**
 - ❖ a single percept may be associated with multiple states
 - ❖ the missing information may distinguish the states

Belief States

- ❖ **the belief-state space consists of all possible states that the agent knows of**
 - ❖ in a fully observable environment, each belief state corresponds to exactly one physical state
- ❖ **the belief-state space contains every possible set of physical states**
 - ❖ exponential with respect to the physical state size
 - ❖ many nodes in belief-state space may be unreachable
 - ❖ they do not correspond to a valid percept/state combination in the physical environment

Search in Belief-State Space

- ❖ **in belief-state space, search is fully observable**
 - ❖ the agent knows its own belief state
 - ❖ there is no sensory input (in belief-state)
- ❖ **the solution is always a sequence of actions for the belief-state**
 - ❖ even if the actual environment is non-deterministic
 - ❖ the percept received after each action is predictable, since it is always empty
- ❖ **the agent updates its belief state as information about the physical states becomes available**
- ❖ **practical approaches are known under various names**
 - ❖ filtering, state estimation
 - ❖ many use probabilistic techniques

Online Search and Unknown Environments

- ❖ **in online search, the agent interleaves search and execution steps**
 - ❖ often necessary in dynamic or non-deterministic environments
- ❖ **in unknown environments, the agent has no choice but to perform online search**
 - ❖ also known as exploration problem
- ❖ **actions may be non-reversible**
 - ❖ this leads to a dead end, where no goal state is reachable
 - ❖ no viable path to the goal
 - ❖ the agent is not necessarily physically “stuck”

Online Local Search

- ❖ **hill-climbing search is an online method**
 - ❖ random restart may not work, however
 - ❖ the agent often can't just be moved to a random point in a real-world environment
- ❖ **random walk**
 - ❖ if the state space is finite, the agent will eventually find a goal or completely explore the state space
- ❖ **combining hill-climbing with memory works better**
 - ❖ store a current best estimate (heuristic) for each visited node
 - ❖ leads to an algorithm called learning real-time A* (LRTA*)
 - ❖ complete for finite, safely explorable environments

Important Concepts and Terms

- ❖ agent
- ❖ AND-OR search tree
- ❖ belief state
- ❖ completeness
- ❖ constraint satisfaction
- ❖ continuous space
- ❖ depth-limited search
- ❖ genetic algorithm
- ❖ general search algorithm
- ❖ goal
- ❖ goal test function
- ❖ gradient descent search
- ❖ heuristics
- ❖ hill-climbing search
- ❖ initial state
- ❖ iterative improvement search
- ❖ local search
- ❖ local beam search
- ❖ non-deterministic actions
- ❖ non-deterministic environments
- ❖ online search
- ❖ operator
- ❖ optimality
- ❖ partially observable environments
- ❖ path
- ❖ path cost function
- ❖ problem
- ❖ search
- ❖ simulated annealing
- ❖ space complexity
- ❖ state
- ❖ state space
- ❖ time complexity
- ❖ uniform-cost search

